FTPMAN Storm

Upgrade bug analysis
Wed, May 9, 2001

While upgrading the Linac front-ends to the new PowerPC-based systems, surprising and catastrophic behavior was encountered relating to the `FTPMAN` protocol support in these systems. This note tries to analyze this bug.

The `FTPMAN` protocol for the continuous mode plotting was upgraded a few years ago, but only recently was the support code for the upgraded version implemented, prompted by insistence that the new Java-based Acnet system only support the newer variations of the `FTPMAN` protocols in front-ends. Specifically, the typecode 2 support was retired and upgraded to typecode 6 support only. The new upgraded implementation considers typecode 2 support to be invalid.

As a review of the scheme for the `FTPMAN` client making requests to the server, consider that the client first issues a "timing" request, using typecode 1, to ask the front-ends about the plot data capabilities supported by the front-end. Two class codes are returned. One identifies the support given for continuous mode plot data, and the other identifies that for the snapshot mode. Specifically for the case used most heavily in Linac, which is 15 Hz plotting, there are two class codes that apply to the continuous mode. Both class codes imply the same capability, except that one indicates that subsequent plot data requests for the indicated devices should be made using the typecode 2; the other indicates typecode 6. After receipt of the reply to the timing request, the client issues its request for the plot data, which is returned repeatedly until the plot request is canceled.

That is the view of data collection by the client that runs in the Acnet consoles. Linac/IRM front-ends add an additional wrinkle to this simple behavior, in that they provide automatic server support for such data requests. When a console sends a data request that includes devices whose data is actually sourced by multiple front-ends, the front-end receiving such a request acts as a server, which means it arranges itself to multicast the same request to the other nodes in its project. Each node that receives the multicast request is expected to reply with data for its own devices. The server, upon receiving such partial replies, places them into proper order and returns a complete reply to the original console client. The client knows nothing of this decomposition and reconstitution arranged by the front-end acting in this server mode.

When entering devices into the Acnet database, a source node for the device must be specified. This source node determines which node receives a data request from an Acnet console. Inside each request is an 8-byte `SSDN` structure with a meaning defined by the front-end software. For the front-ends under discussion here, the `SSDN` structure includes the real front-end node number. By scanning the `SSDN`s for all the devices in the request, then, the front-end can determine whether it must act as a server for the request. Any front-end, in principle, can act as a server for any Acnet data request. In practice, however, all Acnet devices within one project use the same source node; hence, a single front-end is used as a server. (Actually, Linac has always used two such servers.) The original reason for implementing this server-style support was to limit the number of replies that the Vax console must receive. If a data request had to be broken down across 5 nodes, say, the console would have to receive 5 replies. That may not seem so difficult, but for a data request spanning 20 nodes, it could be worse. But another benefit was gained by using the server support. WIth the replies to a data request coming from only a single server node, it made it more likely that the console applications could find Linac data as received to be correlated, meaning that it was all measured on the same 15 Hz cycle. For the 15 Hz Linac, the need for correlated data has always been emphasized as a requirement, especially for Linac studies.

What multicast destination address is used when a request must be multicast by a front-end? Each front-end is configured with a "broadcast node number." To keep the various projects that are supported by this front-end software separate, each project's front-ends use a different multicast address for this purpose. This is especially important for the nodes used as server nodes, as they are the ones most likely to be multicasting requests, at least in the case of Acnet protocol support.

Linac server front-ends have always used `09FC` as the broadcast node number, which is translated into the IP multicast address `0xEF8002FC`, or `239.128.2.252`. Each Linac node, then, must be configured to receive from this multicast address, so that it can "hear" the forwarded requests from the server node.

The Linac upgrade project involves replacement of older 68020-based front-ends with newer ones that are PowerPC-based. The software in the older front-ends is about 2 years old. The `FTPMAN` support, via the `FTPM` local application, is even older; hence, it supports only the typecode 2 continuous mode protocol. The newer systems have recently been upgraded to use the typecode 6 continuous mode protocol. While upgrading each Linac node to the PowerPC version, an implied change in `FTPMAN` support occurs, in which typecode 2 becomes obsolete and invalid for the new nodes that implement support for typecode 6 instead. This is important because an invalid typecode appearing in a received data requests prompts an error reply message that complains about the invalid typecode. In `FTPMAN` parlance, this is a "`15 -1`" error code.

Consider a data request for devices from two upgraded front-ends. The timing reply from the server node will indicate that typecode 6 should be used for requesting continuous mode data. When the typecode 6 request is subsequently received, the server node will multicast it to the rest of the nodes in the project. When this problem was first observed, the server front-end had been mistakenly configured to use `09F9` as the broadcast node number. By convention, this multicast address reaches all Linac/IRM front-ends at Fermilab. (It is common that test front-ends use this broadcast address so as to be able to perform name lookup across all projects.)

When all nodes, perhaps 100 or so, receive the forwarded request, nearly all of them will consider that typecode 6 is invalid, and they will return an error reply. This is a mistake, in that the `FTPMAN` support should not return such an error reply to a multicast message, even though it must return error replies about problems with particular devices included in the request. An invalid typecode error message should be suppressed by a front-end receiving it via multicast, although it should not be suppressed by a front-end that receives the data request individually.

Let us assume that 100 error replies come to the server node, which is a PowerPC node that uses vxWorks. A well-known problem for vxWorks is that its ARP cache size is limited to about 36 entries. But should such reception cause the ARP cache to be filled? Probably not.

Note that it is vital that a front-end be able to determine whether a message is received via multicast or individually. If it thought that a multicast message were addressed only to it, then it might determine that it should be a server for that request, forwarding it to its own broadcast node, which must include itself. This would result in a network storm that could be sufficient to swamp front-ends to such a degree that they might hang up. This is especially true for the slower Linac nodes. If they are completely tied up in network interrupt processing, say, they could miss keeping up with their normal 15 Hz processing, which includes a retriggering of a watchdog hardware circuit on the VME Crate Utility board, which after 100-200 ms of being ignored, enforces a VME crate reset, which prompts the front-end software to reboot.

The old Linac front-ends determine whether a received message was multicast by examining the 6-byte destination physical network address in the frame header. The newer front-ends cannot do this, because vxWorks does not make such low level network information available. We are using a `select()` call to receive from a number of file descriptors to organize all network reception of interest into a single task. When a message arrives, the `recvfrom()` function is called to bring in the datagram. But the return from `select()` does not identify how the message was received that was addressed to a given port. So the new software looks inside the Acnet header to determine the answer to this question. If a request message destination node field is of the form `0x09Fy`, where `y` is any hex digit, then it is considered multicast; otherwise, it is assumed to have been received addressed to the individual node. This convention of multicast node numbers is used by all Acnet support in Linac/IRM front-ends.